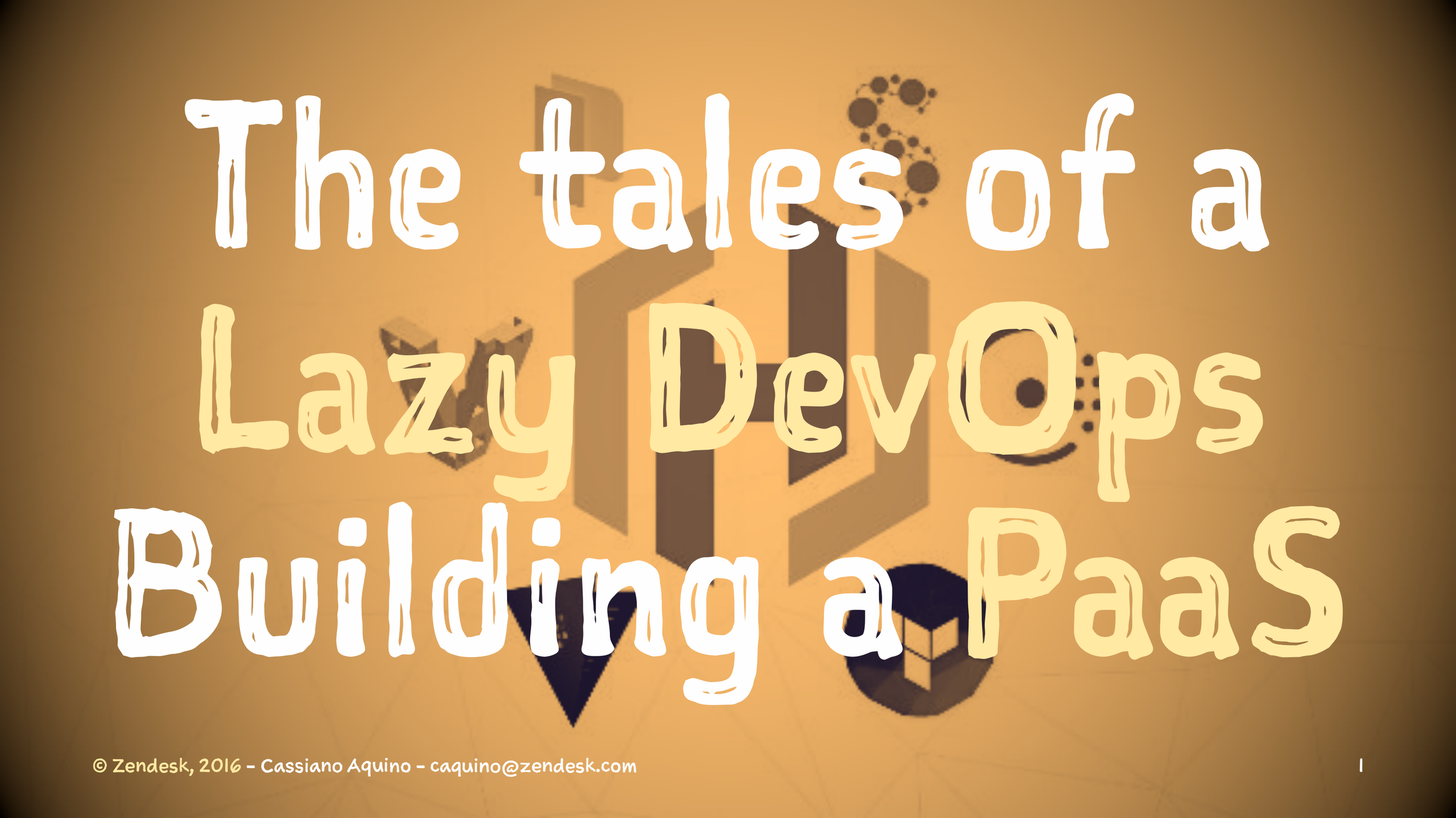


 - Cassiano Aquino - caquino@zendesk.com

Lazy DevOps

Cassiano Aquino
Tech Lead DevOps Engineer @ Zendesk
EI5HPB
@syshero
http://syshero.org/

Once upon a time...

Once upon a time... 2013 Steve Czajka Calligraphy

The legend says that company was acquired and...

Bussines requirements

- SOC2 Compliance
- Secure
- High Availability

BORING

Availability and Security

- Multiple AZs
- Private and Public subnets
- Individual SGs
- CIS Baseline

Just follow AWS guidelines and CIS baseline

SOC2 Compliance

- Audit everything, All changes requires a PR on GitHub
- Deployments using Samson
- Logging, Cloudwatch, and Elasticsearch streaming

even more BORING

Developers requirements

- Minimum impact to the current workflow
- Avoid decreasing developers efficiency
- We don't care about this boring stuff

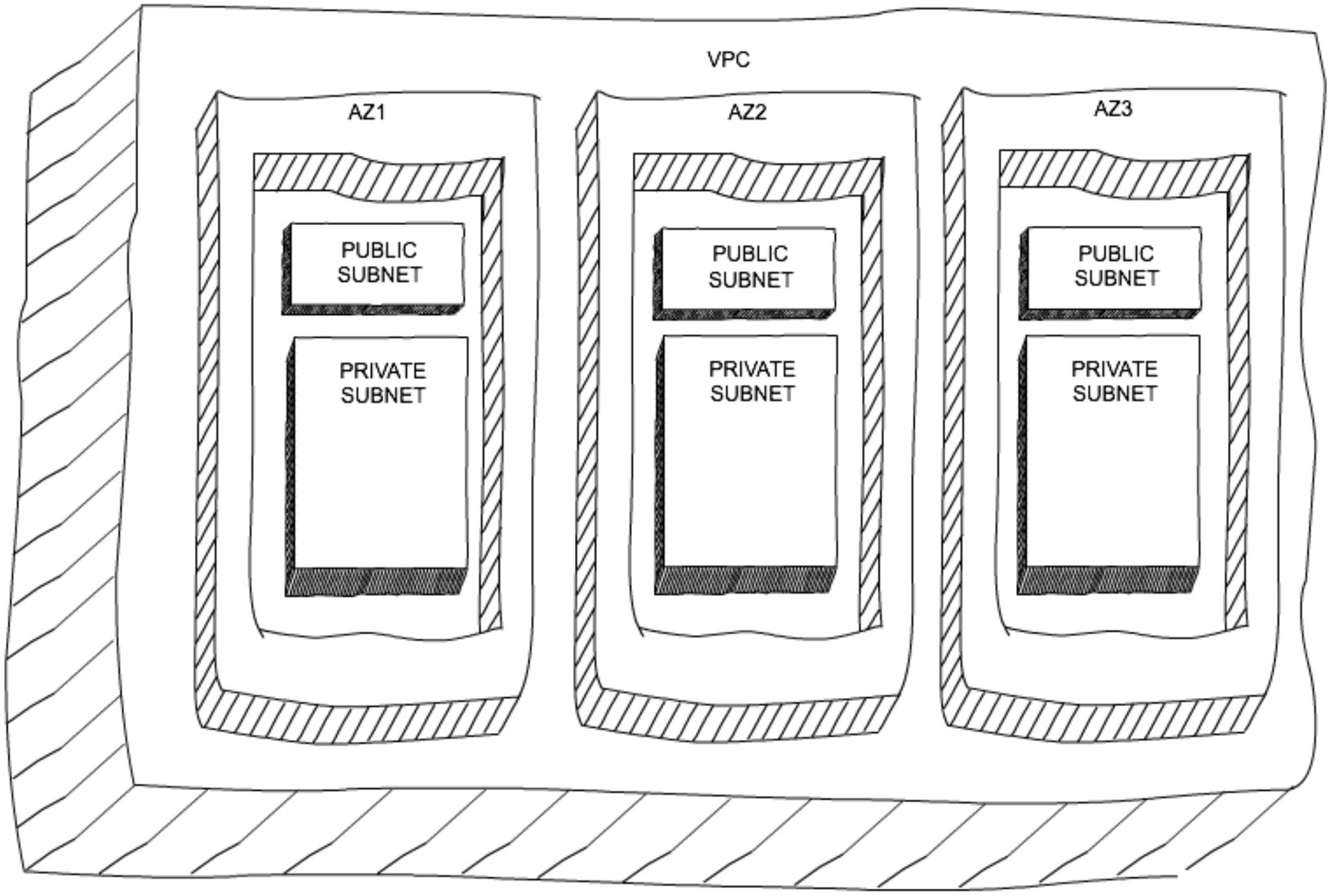
PS: I will try my best, I swear!

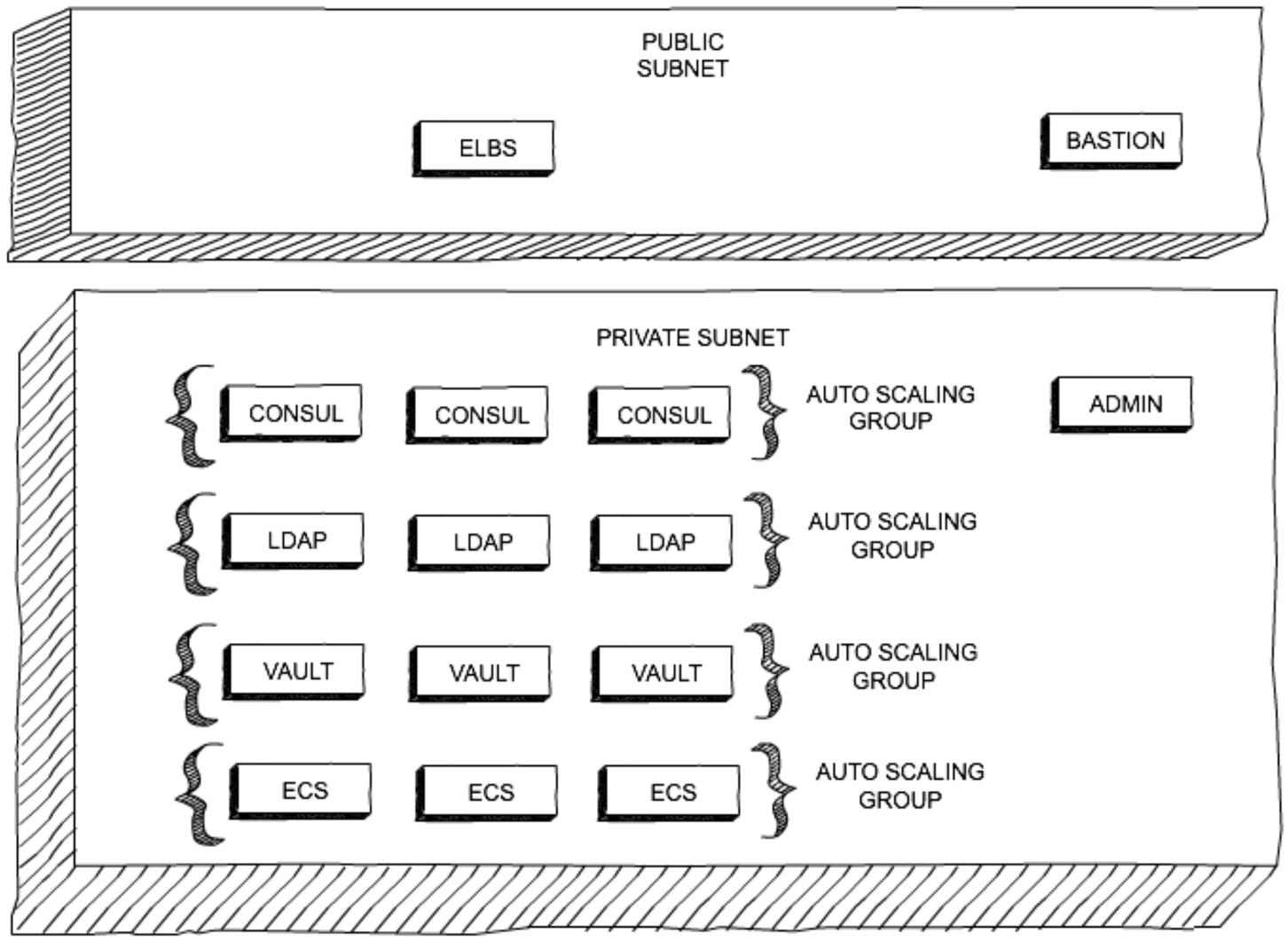

PERSONAL
REQUIREMENTS

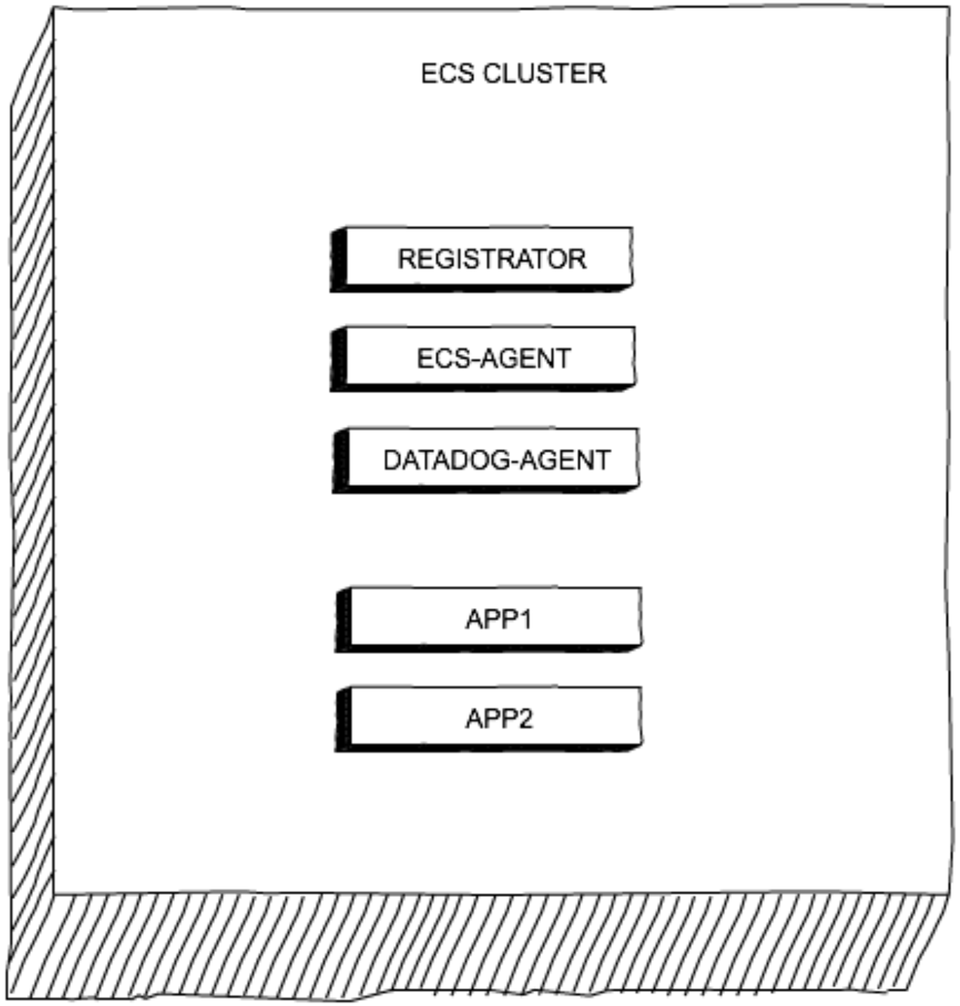
Personal requirements

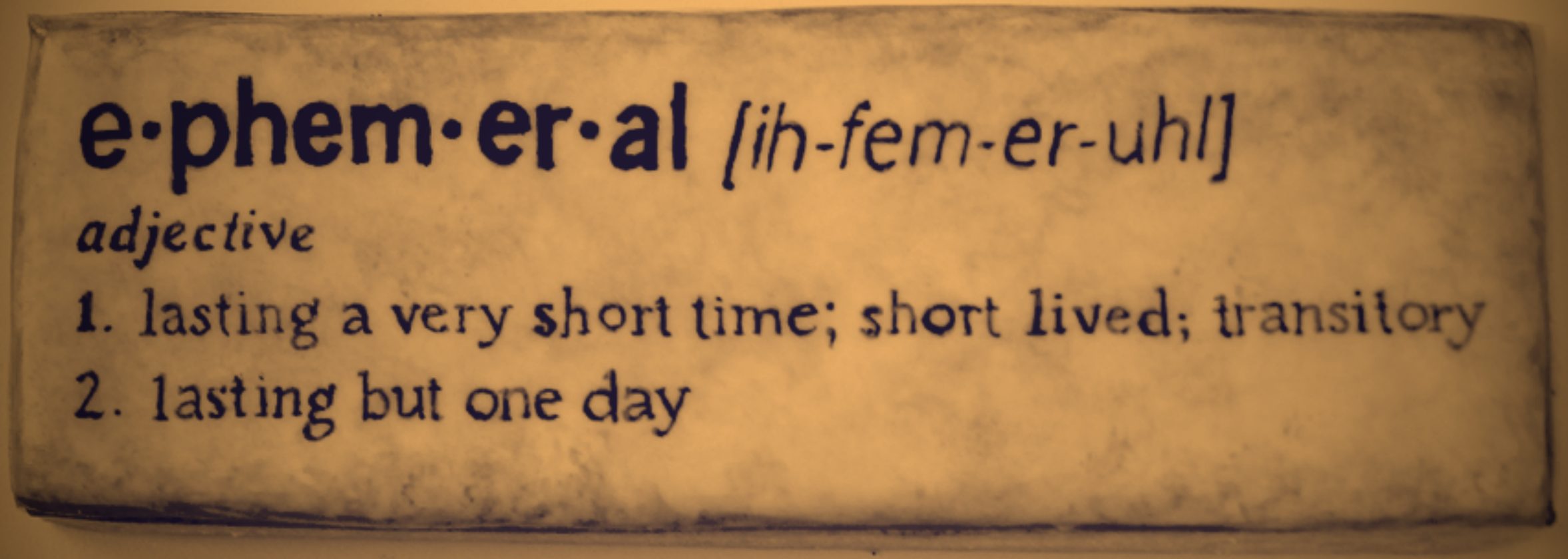
- Ephemeral
- Self-healing
- Minimum work
- Simple

 - Cassiano Aquino - caquino@zendesk.com


VPC
AZ1
AZ2
AZ3
PUBLIC SUBNET
PRIVATE SUBNET
PUBLIC SUBNET
PRIVATE SUBNET
PUBLIC SUBNET
PRIVATE SUBNET


PUBLIC
SUBNET
ELBS
BASTION
PRIVATE SUBNET
CONSUL
CONSUL
CONSUL
AUTO SCALING
GROUP
ADMIN
LDAP
LDAP
LDAP
AUTO SCALING
GROUP
VAULT
VAULT
VAULT
AUTO SCALING
GROUP
ECS
ECS
ECS
AUTO SCALING
GROUP


ECS CLUSTER
REGISTRATOR
ECS-AGENT
DATADOG-AGENT
APP1
APP2


e·phem·er·al [ih-fem-er-uhl]
adjective
1. lasting a very short time; short lived; transitory
2. lasting but one day

Requirements

- No customer data on instances
  - S3
  - RDS
  - Amazon ElastiCache
  - Amazon ElasticSearch


Self
Healing

Requirements

- Automatic configuration
- Automatic application deployment
- Autoscaling Groups

Bootstrap

1. Automation (chef/puppet/ansible/...) ❌
   1.1 Increase complexity
   1.2 Hard to get it right with ASG
2. Custom AMI per hostgroup ❌
   2.1 Hard to maintain and update
   2.2 No individual configuration

Bootstrap

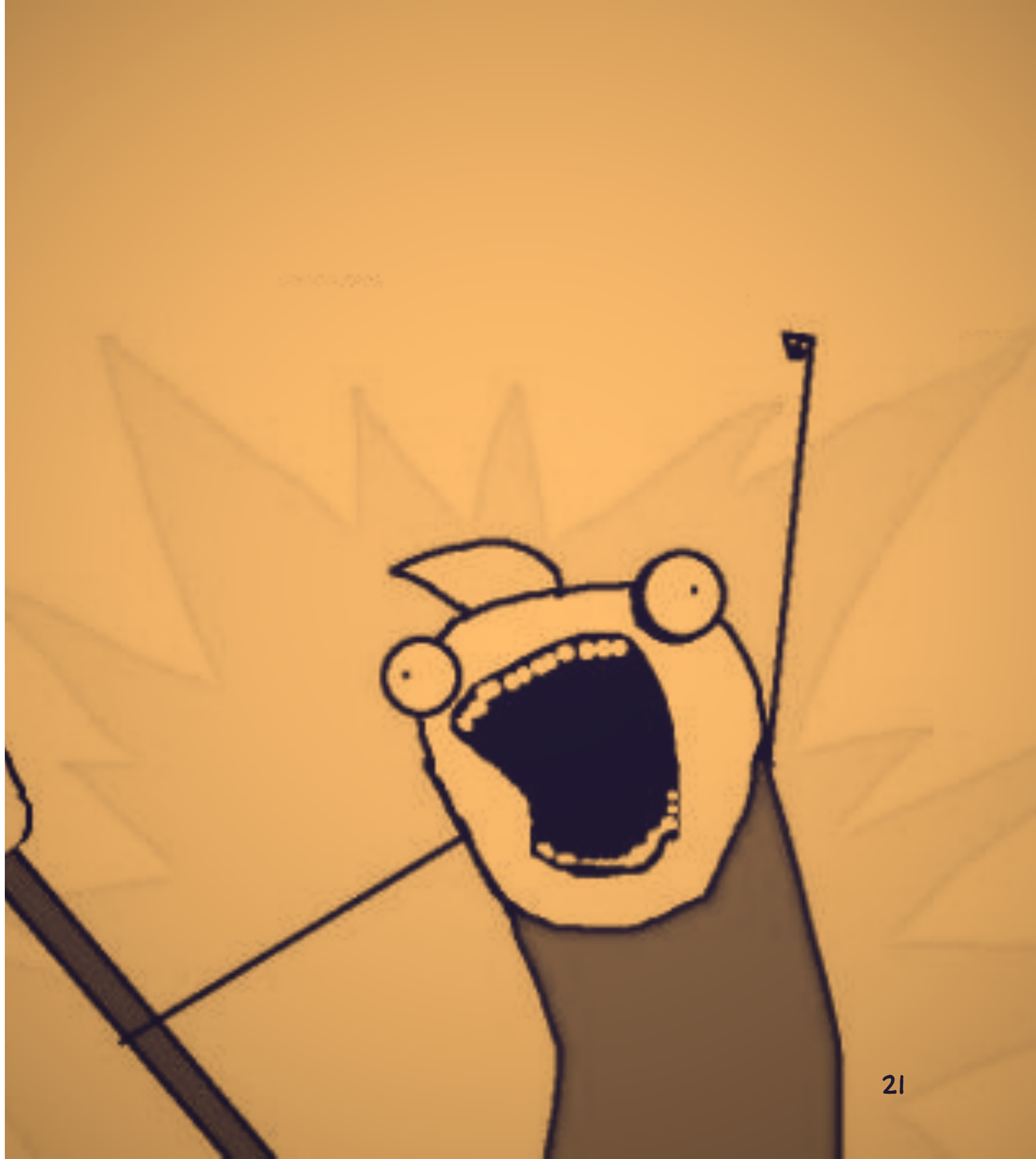
- Base AMI + cloud-config ✔
  - Defaults backed on the AMI
  - Customizations trough cloud-config

Packer

- Example

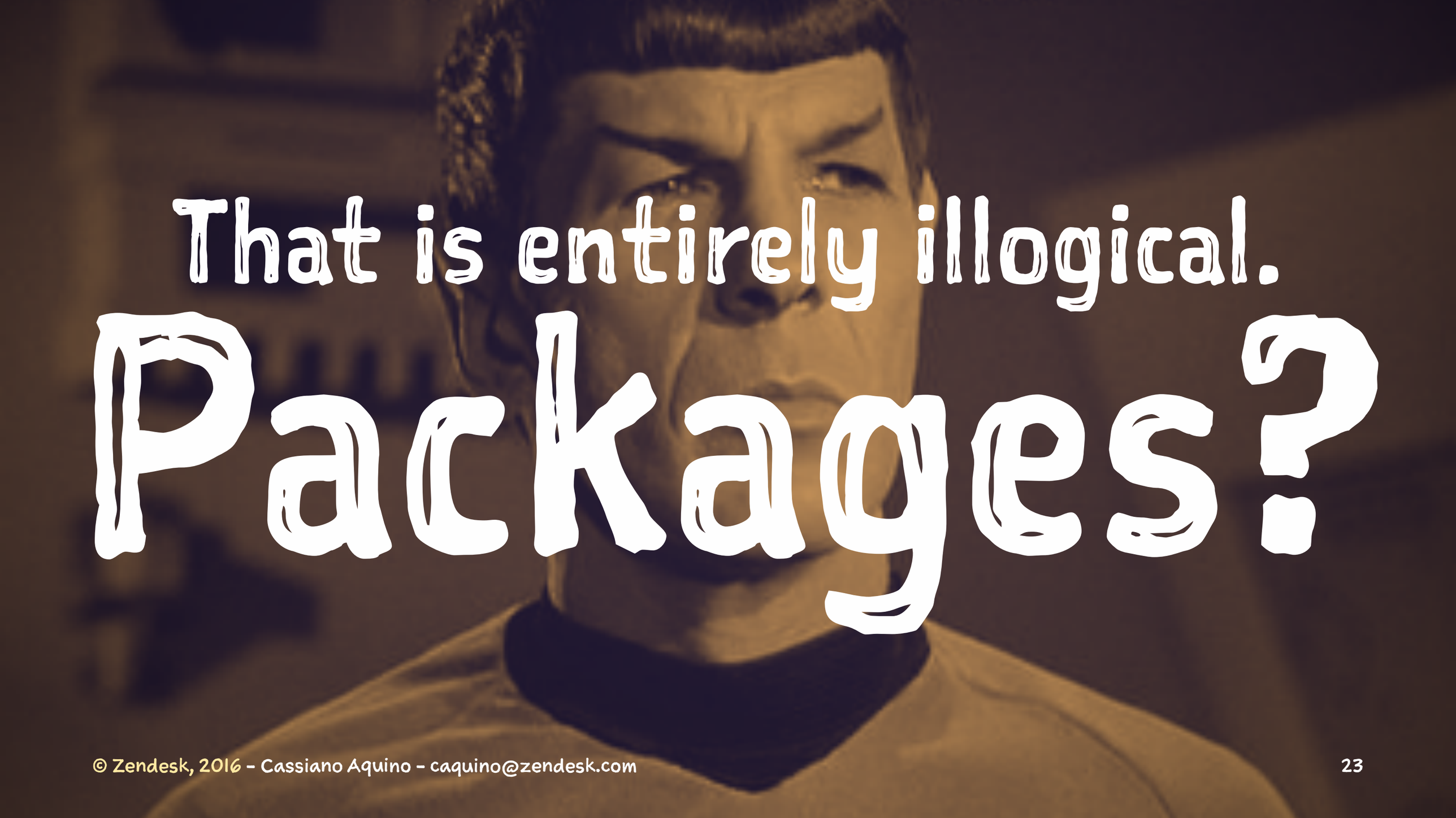
ASG

- Autoscale all the things!
  - Harder to maintain?
  - More Difficult to automate?
  - How to handle new nodes?



Options

- Automation ❌
- Individual Images ❌
- Base image, custom packages, and PMA ✔


That is entirely illogical.
Packages?

Packaging

- fpm-cookery
  - Consul recipe
  - Base config
  - Scripts
- travis-ci
  - Build

OK, BUT WHAT ABOUT REALLY REALLY REALLY CUSTOM CONFIGURATION?

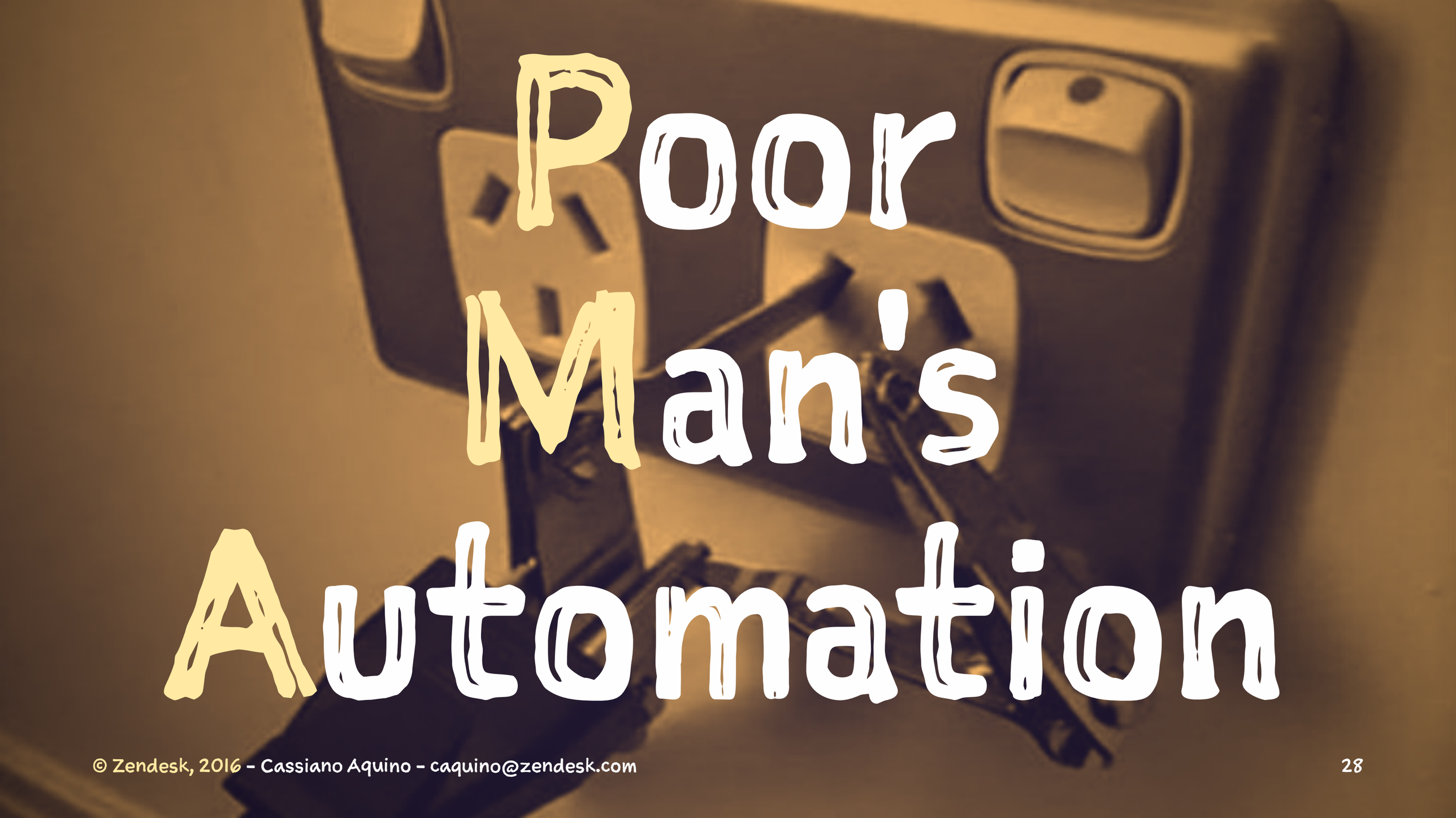
Option 1

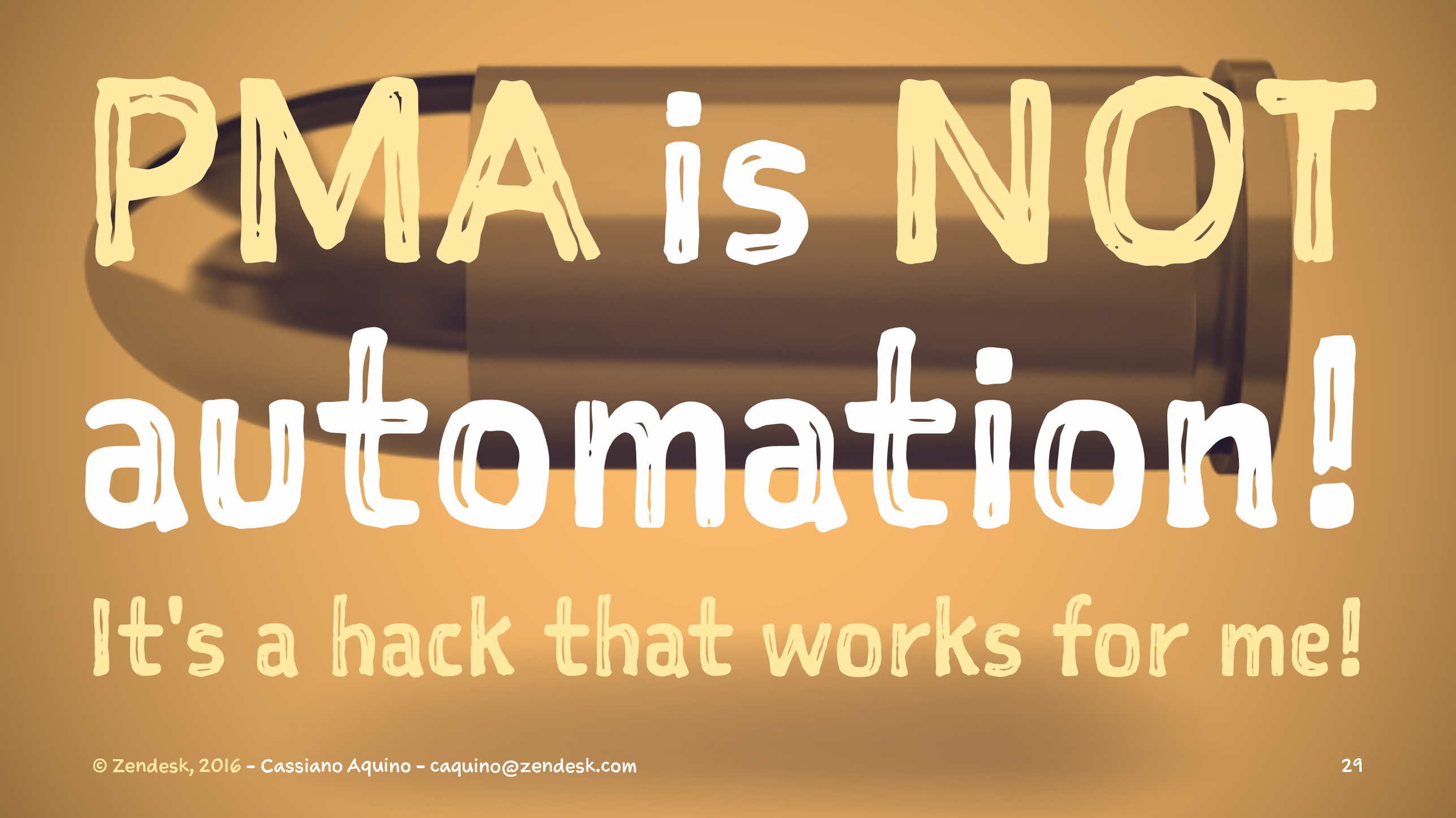
cloud-config

```
#cloud-config
bootcmd:
  - echo 'SERVER_ENVIRONMENT=${environment}' >> /etc/environment
  - echo 'SERVER_GROUP=${name}' >> /etc/environment
  - echo 'SERVER_REGION=${region}' >> /etc/environment
  - echo 'SERVER_HOSTGROUP=${hostgroup}' >> /etc/environment
  - systemctl enable consul-server-config
  - systemctl enable consul-server
runcmd:
  - systemctl start --no-block consul-server
```

cloud-config

- Works well to individual node configuration
- It's not built to handle clusters and groups
- Does not support individual customization on ASGs


Poor
Man's
Automation


PMA is NOT automation!
It's a hack that works for me!

Poor man's automation

- Abusing Systemd
  - Configures Consul / Starts Consul
  - Format EBS / Mount EBS

Infrastructure Configuration

- Terraform
- Packer
- Consul
- Vault

Application Configuration

- Consul - Service Discovery
  - registrator
  - envconsul
  - consul-template
- Vault - Secret Management

ECS

- Service provisioning
- Application deployment


Terraform

Friendly advice

- take good care of your state file
- split your environments in different states
- modules == functions
- terraform fmt
- terraform plan

Conditionals

```
variable "resource_enabled" {
  default = 0
}

resource "aws_instance" "main" {
  count = "${var.resource_enabled}"
  ...
}
```

Terraform

→ travis-ci + terraform validate

Terraform 0.7

```
data "aws_ami" "base_ami" {
  owners      = ["self"]
  most_recent = true
  filter {
    name   = "name"
    values = ["base/*"]
  }
  filter {
    name   = "tag:QAStatus"
    values = ["pass"]
  }
}
```

Terraform 0.7

```
data "aws_iam_policy_document" "consul_ro_policy" {
  statement {
    actions = [
      "ec2:Describe*",
      "autoscaling:Describe*",
    ]
    resources = [
      "*"
    ]
    effect = "Allow"
  }
}
```

Infrastructure and Services

- Both managed by Terraform, different repositories
- Freedom to developers
- Easier to audit
- Incident blast management

Services

```
module "my_awesome_app" {
  source          = "github.com/bimeio/stack//service"
  name            = "my-awesome-app"
  image           = "my_awesome_app"
  port            = 80
  environment     = "${module.stack.environment}"
  cluster         = "${module.stack.cluster}"
  iam_role        = "${module.stack.iam_role}"
  security_groups = "${module.stack.internal_elb}"
  subnet_ids      = "${module.stack.internal_subnets}"
  log_bucket      = "${module.stack.log_bucket_id}"
  zone_id         = "${module.stack.zone_id}"
}
```

What you get?

- an ECS task definition
- an ECS service definition
- a Consul service/health check

External Service

- registrator register service with Consul

NGINX+

```
resolver localhost:53 valid=10s;

upstream backend {
    zone upstream_backend 64k;
    server service.consul service=my_awesome_app resolve;
}

server {
    location / {
        proxy_pass http://backend;
    }
}
```

What about new backends?

→ consul-template

```
{{ range $service := services }}
  {{ if ne .Name "consul" }}
upstream {{ $service.Name }} {
  zone $service.Name 64k;
  server service.consul service=$service.Name resolve;
}
  {{ end }}
{{ end }}
```

I hope we will live happily ever after

THE END